I L 6 N
math

UIUCDCS-R-75-759

no. 759
cop. 2

AN EXPERIMENT ON A UNIFIED CONTROL CONSTRUCT

by

David W. Embley

August 1975

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

UIUCDCS-R-75-759

AN EXPERIMENT ON A UNIFIED CONTROL CONSTRUCT

BY

DAVID W. EMBLEY

August, 1975

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

This work was supported in part by the National Science Foundation under Grant No. US-NSF-EC-41511.

ABSTRACT

This report describes and gives the results of an experiment designed to investigate the psychological soundness of a proposed unified control construct, the KAIL selector. The KAIL selector subsumes several traditional control constructs including _if-then-else_, _case_, and _while_. The experiment compared two sets of control constructs to determine their effect on understanding. One set consisted of the traditional _if-then-else_, _case_, and _while_; the other set consisted of a simplified KAIL selector. Results showed that subjects understood programs better when they were written using the KAIL selector.

ACKNOWLEDGMENTS

Professor Wilfred J. Hansen deserves a special thanks for guiding me through this study. He spent many hours discussing the experiment with me and suggested numerous modifications and improvements.

I appreciate the help given by Professor Richard G. Montanelli on the statistical aspects of this study. He also read the manuscript and offered many helpful comments.

I appreciate the CS201 students who gave their time to take the experiment.

Many others helped in numerous ways. Barr Segal assisted me in preparing data for the statistical package (SOUPAC). Dave Eland and Paul McNabb suggested several improvements for the experiment. Professor George H. Friedman gave technical and administrative assistance, and Kathy Gee typed this report.

TABLE OF CONTENTS

Page

1. INTRODUCTION 1

1.1 Objective 1

1.2 KAIL 1

1.3 Related Work 2

2. THE KAIL SELECTOR 3

2.1 Basic Selector 3

2.2 Defaults 5

2.3 Else 5

2.4 Sel-tags 6

2.5 Iteration 6

2.6 Actions 7

2.7 Blocks and Expressions 8

3. THE EXPERIMENT 9

3.1 Hypotheses 9

3.2 Subjects 11

3.3 Teaching the Languages 14

3.4 Problems and Programs 16

3.5 Question-Answer Session 17

4. RESULTS 22

4.1 Observable Measures of Understanding 22

4.2 Difficulties 22

4.3 Background 23

4.4 Performance Statistics 23

Page

4.5 Learning Statistics 26

4.6 Problem Comparison 26

4.7 Correlations 28

5. CONCULSION 32

5.1 Summary of Results 32

5.2 Suggestions for Further Research 32

REFERENCES 34

APPENDIX A. FULL GRAMMAR OF THE KAIL SELECTOR 35

APPENDIX B. SUBJECT BACKGROUND DATA 36

APPENDIX C. PROBLEMS AND PROGRAM SOLUTIONS 39

APPENDIX D. QUESTIONS 44

APPENDIX E. PERFORMANCE STATISTICS 46

APPENDIX F. CORRELATIONS 53

LIST OF TABLES

Table		Page
1	Group Averages	25
2	Self-evaluations	27
3	Average Learning and Reviewing Times	27
4	Correlations with Correct Answer Statistics	29
5	Data Correlated with Class Standard Score	30

LIST OF FIGURES

Figure Page

1 Design Layout 10

2 Introduction to the Experiment 12

3 Experiment Overview 13

4 Instructions 18

5 Subjective Self-evaluation 18

6 Preparation for Question-Answer Session 19

7 Possible Requests for Help 19

8 Typical Question 20

1. INTRODUCTION

1.1 Objective

Historically, programming languages have been conceived and developed by individuals or relatively small groups [1]. Language features have arisen from the designer's own experiences, but little or no effort has been made to determine if the design is psychologically sound [2].

This report describes an experiment to investigate the psychological soundness of a new control construct called the KAIL selector [3]. The selector (outlined in chapter 2) is the dominant control construct of the KAIL programming language [4].

1.2 KAIL

This work is part of the Automated Computer Science Education System (ACSES) project [5] which uses the PLATO CAI system [6] to teach elementary computer science. PLATO runs on a CDC Cyber 73 dual processor with two million 60-bit words of extended core storage. Attached to the processor are over nine hundred plasma display terminals, which were invented by the PLATO group for this project. The peak comfortable system load is about four hundred simultaneous users.

Under PLATO, all programs are written in a special system language called TUTOR [7] which mixes FORTRAN-like flow of control with peculiar CAI sequencing and sophisticated answer judging facilities. The language KAIL is an attempt to influence the development of TUTOR by showing that modern control constructs can be combined with the higher level features of TUTOR.

1.3 Related Work

In 1971, Gerald M. Weinberg wrote The Psychology of Computer Programming [8]. Its stated purpose was "to trigger the beginning of a new field of study: computer programming as a human activity." In the book, he advocated the use of experimentation as a method to study programming and suggested that human factors be considered in language design.

Since the appearance of Weinberg's book, several researchers have conducted psychological experiments on programming language features. Using inexperienced programmers, three psychologists concluded that nested if programs are easier to understand than corresponding goto programs [9]. Larry Weissman listed and catagorized factors affecting the complexity of programs, began development of a methodology for studying the psychological complexity of computer programs, and conducted some experiments attempting to determine which factors or combinations of factors reduce complexity [10, 11]. At the University of Indiana, memorization tests were used to measure the effect of program structure on understanding [12]. Recently, John D. Gannon completed a Ph.D. thesis for which he gathered empirical evidence to support or discredit specific language design decisions [13].

Like Gannon's work, the KAIL selector experiment was an approach to language design. It examined, however, a new syntatic and semantic construct rather than a controversy between existing features. The methodology of this experiment resembled Weissman's. One major difference was the use of automated rather than manual techniques. Because the PLATO CAI system administered the experiment on-line, the language constructs could be taught as part of the experiment; and precise timing data and all other relevant information could be gathered.

2. THE KAIL SELECTOR

2.1 Basic Selector

The KAIL selector differs from most typical high level control constructs in three ways:

1. It combines flow of control with "answer judging," the process by which a Computer-Aided Instruction (CAI) program determines whether a student reply matches one of the replies anticipated by the lesson author.
2. It combines most of the other control constructs that have been widely used, e.g., if-then-else, case, cond, while, ...
3. The syntax suggests an answer to the trivial, yet thorny, question of how to end a nested block of code. Previous proposals include: end of one statement, end, fi-esac-elihw ..., endif-endcase-endwhile ... .

This KAIL selector is the result of a search for a syntactic structure for CAI answer judging. In answer judging the common control operation is to compare a student response against a set of expected correct and incorrect responses. This suggests a control syntax where a given expression (usually the student's reply) is tested against a number of alternatives. The full selector syntax is given in appendix A, but will be introduced piecemeal in the sections below.

The essential selector syntax can be sketched as

```
selector : "[" statement-sequence control choices "]"
~~~~~~~~       ~~~~~~~~~ ~~~~~~~~ ~~~~~~~ ~~~~~~~

control  : "if" expression
~~~~~~~         ~~~~~~~~~~
```

```
choices  : λ
         | "|"  relation  expression  ":"  statement-sequence  choices
```

where expression, relation, and statement-sequence are defined by the base language. (The two expressions are referred to as control-expression and choice-expression, respectively.) The selector is executed in four steps:

1. Execute the statement-sequence in the selector.
2. Evaluate the control-expression and save its value in a temporary, t.
3. Test each choice in turn until a "selected choice" is found such that

   (t relation choice-expression) yields true.
4. Execute the statement-sequence in the selected choice.

If no selected choice is found, step four is omitted.

This simple example tests a student's arithmetic abilities:

```
x := rand(25); y := rand(25); comment set x and y to random integers
                                   in [1,25];
write What is <(x)>  +  <(y)>? ;
[accept reply; if reply
   | = x+y:  write Very good; right_count := right_count + 1;
   | = x*y:  write Add, don't multiply;
   | > x+y+20:  tx := x div 10; ty := y div 10;
                write But <(tx+1)>0 + <(ty+1)>0 is only <(tx+ty+2)>0!;
   ];
```

2.2 Defaults

The key to the adaptability of the KAIL selector is its default values. The expressions and relations may be omitted with the following result:

control-expression		true	
relation		=	
choice-expression	(1)	true	
choice-expression	(2)	false	
choice-expression	(i)	i-2	(i ≥ 3)

(In TUTOR, -1 is true, and 0 is false. In a spirit of compatibility, KAIL has adopted these conventions.) When the choice-expression is omitted, the relation and ":" must also be left out.

By proper choice of default values, the selector has the semantics of several other control constructs, including: if-then-else, labelled-case (similar to LISP's SELECT), unlabelled-case, and COND. For example, for if-then-else there are only two choices and no choice-expressions (e.g., [if x ≥ y | max :=x | max :=y]). For a labelled-case, the relations are omitted and the choice-expressions are constants. (It would be closer to a labelled-case if the quotes could be omitted from string constants, but then the selector could not be mapped into such a variety of constructs.)

2.3 Else

One additional choice is

choice : " | else : " statement-sequence

If evaluation reaches this choice, it is always the "selected choice" and its statement-sequence is executed. This corresponds to the out clause in the ALGOL 68 case construct. Notice the distinction between "|" and "| else:" in an if-then-else: without the else, the second choice will only be selected if the control-expression yields false; with the else, any value but true will select the second choice.

2.4 Sel-tags

Selectors can become deeply nested. To help the reader and compiler match brackets, the programmer may tag them with matching sel-tags:

selector : sel-tag "_" selector "_" sel-tag

where the two sel-tags are both the same unique identifier. In practice, the separation of the tag from the bracket by the unobtrusive underline has a reasonably good appearance. (Intuitively, this is so because bracketed expressions are often surrounded by spaces.)

2.5 Iteration

The KAIL selector unifies iteration and selection.

selector : "*[" statement-sequence [loop-control choices] "]*"

loop-control : "while" expression | "until" expression

For iteration, a fifth step is added to the selector semantics:

5. For while, if no "selected choice" is found, exit; otherwise, return to step one. For until, if no "selected choice" is found, return to step one; otherwise, exit.

Notice that "*" distinguishes iteration from selection both at the beginning and ending of a loop, and that the keywords, while and until reinforce this notion from within the construct.

Just as the basic selector maps into several branching constructs, the iteration construct maps into several existing loop constructs, including: while, repeat-until, loop-while-repeat [14], and the nesting of a single case statement inside of a while. For example, for while, there is only one choice, and either the control-expression, the choice-expression, or an appropriate combination serves as the predicate. For repeat-until, the body of the loop is the initial statement-sequence, and no choice appears.

As an example, consider the inner loop of Hoare's "Quicksort" algorithm [15].

```
comment m < n and A(m-1) exists and A(m-1) < A(i) for m ≤ i ≤ n;
i := m;  j := n-1;  v := A(n);
*[    *[ while | A(i) < v : i := i+1; ]*; comment incr i;
      *[ while | A(j) > v : j := j-1; ]*; comment decr j;
while i < j |
       A(i) :=: A(j);  comment swap A(i) and A(j);
]*;
```

This example illustrates how the selector solves the problem of looping "n and a half times" [14] without introducing an exit or a goto. Such problems arise often in practive, and sometimes almost demand a goto. Goto-free programmers usually resort to code duplication and/or redundant tests.

2.6 Actions

Actions alter normal flow of control patterns.

```
  choice  :  "|" [relation  expression  ":"] statement-sequence [action]
  action  :  "exit" [sel-tag] | "again" [sel-tag]
```

Exit causes control to leave the selector; again directs control to the beginning of the initial statement-sequence. Sel-tags allow exit and again from within nested selectors.

Actions may be used in the CAI example of section 2.1 to force the student to answer correctly before proceeding. A programmer may use selection and again, but iteration and exit would serve just as well:

```
x := rand(25);  y := rand(25);
write What is <(x)> + <(y)>? ;
[ accept reply;  if reply
    | x+y : write Very good;  right_count := right_count+1;
    | x*y : write Add, don't multiply; again;
    | >x+y+20 : tx := x div 10;  ty := y div 10;
                write But <(x+1)>0 + <(ty+1)>0 is only <(tx+ty+2)>0! ;
                again;
];
```

2.7 Blocks and Expressions

If both the control and the choice-sequences are omitted, the selector groups statements into a block.

The selector may be used in an expression context by replacing the statement-sequence in every choice with an expression. There must be at least one choice. Note that this construct permits statements inside expressions.

3. THE EXPERIMENT

3.1 Hypotheses

Observations on the characteristics of the KAIL-selector and its use in programming lead to the hypothesis that the KAIL selector is easier to understand than an equivalent set of traditional constructs. The selector unifies selection and iteration and subsumes CAI answer judging and most typical high level control constructs. This should enhance understanding because programmers would only need to know one control construct, not several. In the selector snytax, symbols replace keywords in some instances. This should also enhance programmer understanding because it tightens the code and packs more information into a program listing. For instance, the closure itself indicates whether or not the statement is to be repeated.

In order to test the hypothesis, it is first necessary to devise a method for measuring understanding. A subject's ability to comprehend is a function of several factors. These include the following:

Environmental factors:

1. Educational experience
2. Knowledge of programming languages
3. Basic intelligence
4. Motivation

Observable factors:

1. Speed at answering questions
2. Accuracy at answering questions
3. Self-evaluation of understanding

Together, these factors comprise a model of subject understanding.

This model of understanding allows the hypothesis to be broken down into measurable variables. The effects of environmental factors can be minimized by grouping subjects so that among groups, background and motivation are similar. The hypothesis can then be tested by measuring the observable factors. If the hypothesis is correct, subjects working with the selector should be able to answer more questions and answer them quicker; they should make fewer mistakes; and they should give higher self-evaluations.

To test these hypotheses an experiment was devised. Two versions of two programs were written: one version in KAIL, one in a language with standard control constructs. Each subject was assigned to one of four groups as shown in figure 1.

		language	
program order	#1, #2	group #1	group #2
	#2, #1	group #3	group #4

Figure 1. Design Layout

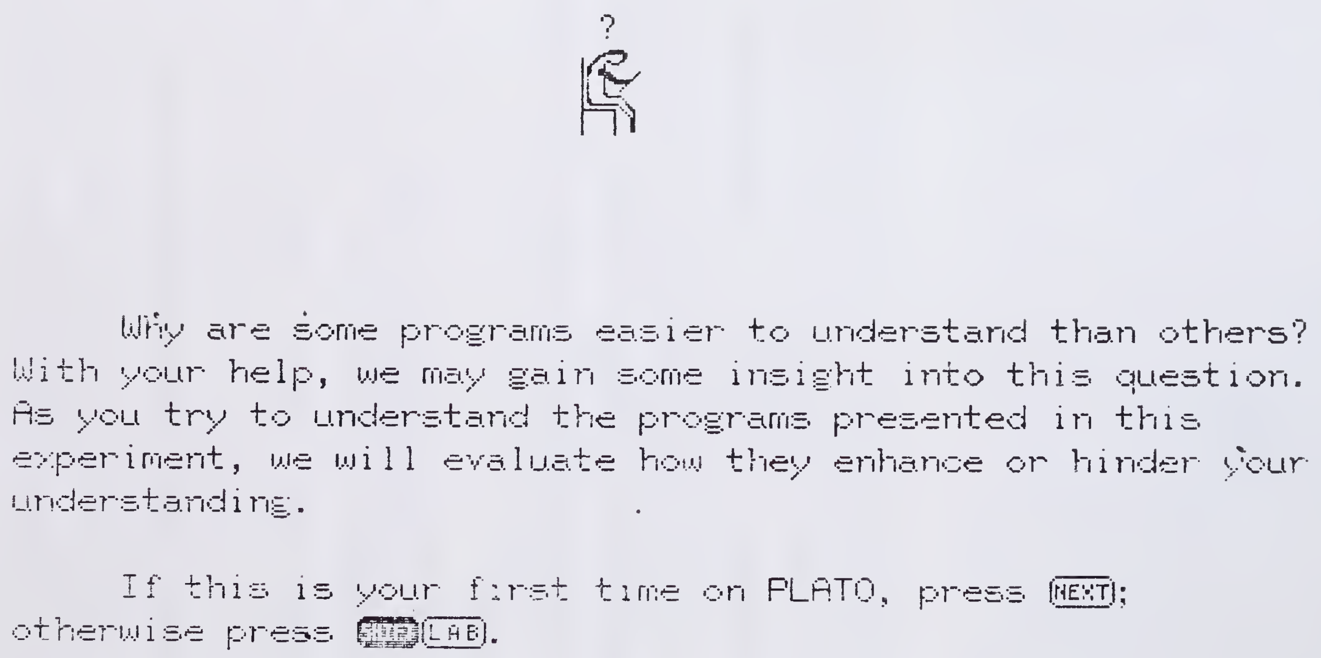
3.2 Subjects

The subjects were volunteers from CS 201, Spring Semester, 1975. Most were sophomores or juniors and had previously taken one programming course using FORTRAN, PL/1, or both. The typical subject's programming ability consisted of experience in one or two languages and some knowledge of a smattering of others. About half had previously used PLATO, some quite extensively. The experiment gathered this background information along with each subject's name and social security number which was later used to associate a student's CS 201 grade with the data. Appendix B contains a table which summarizes this information.

To motivate subject participation, it was explained to a general CS 201 class session that an experiment was being conducted on PLATO to determine why some programming languages are easier to understand than others, and it was pointed out that this would be an opportunity to become acquainted with the PLATO system and participate in an unusual experiment. Fifty-three percent of the CS 201 students eventually responded and came to one of several sessions when terminals were reserved.

In each session as the actual experiment began, this announcement was made, "You will not be graded on how well you do; however, please treat this experiment like an exam." After subjects signed in, the system welcomed them and encouraged them to do their best as illustrated in Figure 2.

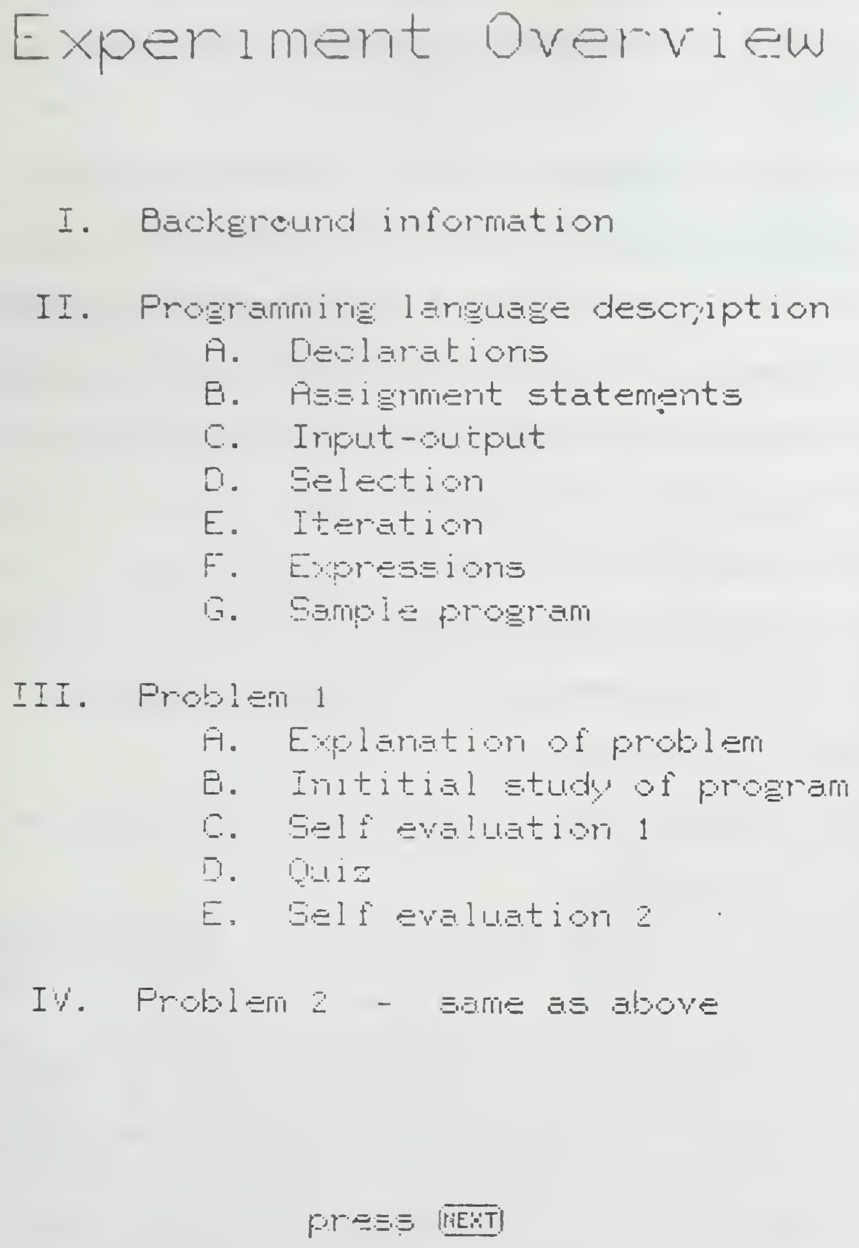
If a subject was not acquainted with PLATO, the system taught the use of function keys and other basics. (Since the experiment only required short responses, often only a single keypress, extensive PLATO experience was unnecessary.)



Why are some programs easier to understand than others? With your help, we may gain some insight into this question. As you try to understand the programs presented in this experiment, we will evaluate how they enhance or hinder your understanding.

If this is your first time on PLATO, press NEXT; otherwise press SHIFT LAB.

Figure 2. Introduction to the Experiment



Experiment Overview

I. Background information

II. Programming language description
   A. Declarations
   B. Assignment statements
   C. Input-output
   D. Selection
   E. Iteration
   F. Expressions
   G. Sample program

III. Problem 1
   A. Explanation of problem
   B. Inititial study of program
   C. Self evaluation 1
   D. Quiz
   E. Self evaluation 2

IV. Problem 2 - same as above

press NEXT

Figure 3. Experiment Overview

The system then outlined the experiment for all subjects (Figure 3).

After subjects completed the background information section, the experiment assigned each to one of the four cells by first computing a level number

level number = f(education level, programming experience)

and then assigning subjects on the same level successively to cell 1, 2, 3, 4, 1, 2, ... , etc. (As it turned out, f was unnecessarily complicated because the students in CS 201 were already reasonably similar. Even worse, f divided the subjects unevenly among the four cells. Successively assigning subjects to cells would have been sufficient.)

3.3 Teaching the Languages

The two objectives of the instruction section of the experiment were to teach the syntax and semantics of the assigned language and to establish some minimum level of understanding.

Declarations, assignment statements, and input-output statements were identical in both languages.

```
declaration  :  integer identifier-list  |  character identifier-list
~~~~~~~~~~~             ~~~~~~~~~~~~~~~                ~~~~~~~~~~~~~~~
assignment-statement  :  variable ⇐ expression
~~~~~~~~~~~~~~~~~~~~     ~~~~~~~~   ~~~~~~~~~~
input-output-statement  :  read identifier-list  |  write identifier-list
~~~~~~~~~~~~~~~~~~~~~~          ~~~~~~~~~~~~~~~           ~~~~~~~~~~~~~~~
identifier-list  :  identifier  |  identifier, identifier-list
~~~~~~~~~~~~~~~     ~~~~~~~~~~     ~~~~~~~~~~  ~~~~~~~~~~~~~~~
```

Selection, iteration, and expressions differed in the two languages. The selector syntax chosen for use in the KAIL-like language emphasized the basic selector characteristics. The other language was akin to ALGOL 68. Keywords were used liberally, and selection and iteration were different constructs. Also, using the ALGOL 68 closures fi, esac, and od made the introduction of begin and end unnecessary.

In the ALGOL-like language selection had two forms:

```
selection  :  if-statement  |  case-statement
if-statement  :  "if" integer-expression "then" statement-sequence
                 "else"  statement-sequence "fi"
case-statement  :  "case"  integer-expression  choices  "esac"
choices  :  ":"  [ subscript ]  statement-sequence
          | ":"  [ subscript ] statement-sequence  choices
```

For the if-statement, if the integer-expression evaluated to 0, the then statement-sequence was executed; if it evaluated to 1, the else statement-sequence was executed; otherwise, neither statement-sequence was executed. (In both languages, boolean expressions evaluated to 0 if true and 1 if false.) For the case-statement, the integer-expression was evaluated and saved in a temproary t; then the statement-sequence in the t+1st choice was executed. If t+1 was out of range, no statement-sequence was executed. For example,

case k $:_0$ write "a"; $:_1$ write "b"; $:_2$ write "c"; esac;

output "a" if k is 0, "b" if it is 1, "c" if it is 2,and nothing if k is anything else.

In the KAIL-like language,selection had one form:

```
selection  :  "[" integer-expression  choices "]"
choices  :  "|"  [ subscript ]  statement-sequence  choices
```

When executed, the integer-expression was evaluated and saved in a temporary t, then the statement-sequence in the t+1st choice was executed. If t+1 was out of range, no statement-sequence was executed. In this language, iteration appeared in exactly the same form as selection except that "*" was placed in front of the left bracket and after the right bracket to indicate repetition.

```
iteration  :  * selection *
```

The semantics of iteration were identical to the semantics of selection except that as long as t+1 was in range, the statement was repeatedly executed. For example,

```
sum ⇐ Ø;   i ⇐ 1;
*[ i≤n ▌ sum ⇐ sum + i;   i ⇐ i + 1; ]*;
```

summed the digits from 1 to n.

In the ALGOL-like language, iteration had the form:

iteration : "while" integer-expression "do" statement-sequence "od"

The statement-sequence was repeatedly executed as long as the integer-expression was true (had value 0).

In both languages, simple expressions were identical, but selection expressions were different. These had the same form as selection statements only the statement-sequences were replaced by expressions.

To establish some minimum level of understanding, the experiment asked each subject six questions as it taught the language. Each question tested some aspect of the language; moreover, the questions served to familiarize subjects with the type of question-answer interaction required later in the experiment.

3.4 Problems and Programs

The two problems were chosen for the experiment such that the programs would exemplify all important variations of the language constructs. One problem was to find all automorphic numbers less than or equal to a number read at the beginning of the program. The solution to this problem read naturally in the ALGOL-like language because it made common use of if-then-else, while, and case. The other program analyzed arithmetic expressions involving the symbols "+", "-", and "a". The solution to this problem took advantage of the selector by eliminating a level of syntactic structure. Appendix C gives the description of these problems and the four programs used in the experiment.

It is important to point out that one program was designed to favor the ALGOL-like language group and the other to favor the KAIL-like language group. This leads to a further hypothesis: the group using the KAIL-like language taking the arithmetic expression problem last should perform best. This is because they would have the advantage of working the program which maximizes the used of the selector after becoming familiar with the KAIL-like language.

3.5 Question-Answer Session

After describing one of the problems, the experiment proceeded with the instructions shown in Figure 4.

A subject studied the program for 100 seconds and then made a subjective self-evaluation as shown in Figure 5.

The experiment then prepared the subject for the question-answer session and encouraged good performance (Figure 6).

All subjects answered identical questions. The 10 questions for each problem appear in appendix D. Before each question, the subject could review by pressing "help", "shift-help", or "data" as shown in Figure 7. On pressing the helpkey, a brief six-line explanation appeared on the bottom of the page; the program remained above. By pressing "shift-help", the subject gained access to the instructional material used in teaching the language. On pressing "data", the problem explanation reappeared.

When the subject pressed "shift-next", Figure 8 shows how the system presented the next question along with a reminder to work quickly.

The subject either answered the question correctly or gave up after 2 minutes, or 3, or 3 1/2, or 3 3/4, etc., up to 4 minutes. The system noted the elapsed time for each input and recorded all wrong answers.

When you press SHIFT NEXT, the program will appear. After 1ØØ seconds, it will disappear; and you will be asked to evaluate yourself on how well you think you understand it.

This is not the last time you will see the program, so your only task at present is to understand how the the program works. For your convenience, you will be told how much time is left about every 1Ø seconds.

Figure 4. Instructions

Enter the number which best describes your feelings about how well you understand the program at this time.

Ø. I don't understand it at all.
1.
2.
3. I get the gist of how it works.
4.
5.
6. I can explain how it works.
7.
8.
9. I understand it perfectly.

The numbers which are not labeled are meant to be gradations of understanding in between those which are labeled.

>

Figure 5. Subjective Self-evaluation

Quiz time! There are 10 questions. They will appear one at a time on the same page as the program. Before each question, you will have the chance to review the language and the problem description; but while you are answering a question, no help will be available.

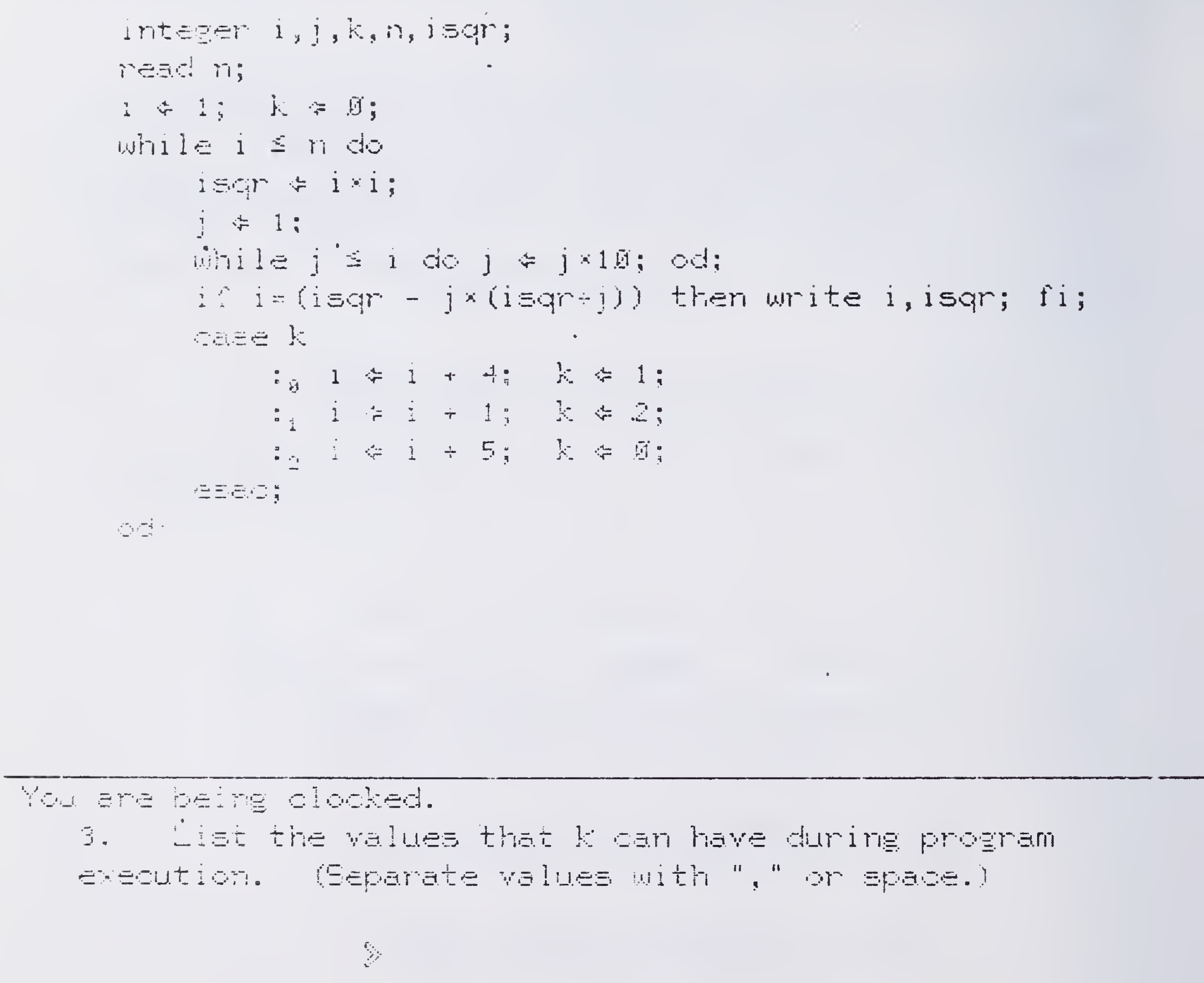
The objective is to answer the question as quickly as possible. After 2 minutes, you will be given the opportunity to give up or continue on for another minute or so. Answer the question to the to the best of your ability because your afforts will help us learn about the psychology of programming.

Figure 6. Preparation for Question-Answer Session

```
HELP  quick review      SHIFT HELP  extensive review
DATA  to see the program explanation again
SHIFT NEXT  for a question about this program

Note:  You will not be able to request
       help while answering questions.
```

Figure 7. Possible Requests for Help



```
integer i,j,k,n,isqr;
read n;
i ← 1;  k ← Ø;
while i ≤ n do
     isqr ← i×i;
     j ← 1;
     while j ≤ i do j ← j×1Ø; od;
     if i=(isqr - j×(isqr÷j)) then write i,isqr; fi;
     case k
          :Ø  i ← i + 4;  k ← 1;
          :1  i ← i + 1;  k ← 2;
          :2  i ← i + 5;  k ← Ø;
     esac;
od;
```

```
You are being clocked.
   3.   List the values that k can have during program
   execution.  (Separate values with "," or space.)

                    »
```

Figure 8. Typical Question

After completing question 10, the subject made a final self-evaluation on the 0 - 9 scale as before. The second problem was then presented in the same fashion as the first, and the experiment was complete.

4. RESULTS

4.1 Observable Measures of Understanding

As explained in section 3.1, the effect on understanding was to be measured in several ways:

1. number of questions answered correctly
2. number of questions answered correctly on first attempt
3. average time taken to answer a question correctly
4. initial self-evaluation
5. final self-evaluation
6. difference between final and initial self-evaluation

These performance statistics are summarized in appendix E, and the important results are discussed below.

The experiment also gathered statistics on the amount of time each subject spent in learning and in reviewing the assigned language and in studying the problem descriptions. These results are mentioned in the text below. Correlations among performance data, language knowledge, CS 201 grade, and the amount of time spent learning the language are found in appendix F and discussed below.

4.2 Difficulties

Several difficulties were encountered during the experiment. Some of these were expected; others were unexpected.

Since the PLATO CAI system can only judge answers correctly if they are anticipated by a lesson author (e.g., 10000 and 1,000 are different), the experiment recorded all answers to every question, whether right or wrong, and logged the appropriate elapsed time. As it turned out, few subjects entered responses which were judged incorrectly. Typing in "_" (underline) for "-" (minus) was the most prevalent of these few errors. In these cases, the raw data was updated to reflect the appropriate system logging as if these answers had been anticipated.

Partial data from two subjects was lost because of a buffer overflow in the monitoring program. This bug was discovered and fixed before most of the subjects took the experiment.

During one session, the system crashed. Of the sixteen subjects in this session, five completed the experiment before the crash, nine completed one of the two problems, and two had not yet completed the first problem. Of the nine who completed only one problem, four were using the KAIL-like language and five were using the ALGOL-like language. In addition to the system crash, at another session, two subjects encountered hardware screen-overwrite problems with their terminals causing some of their data to be declared invalid. Mostly, this partial data was ignored, but it was included in the analysis when theoretically appropriate.

4.3 Background

No significant differences appeared in any background data. Appendix B gives the background of the average subject in each cell for all factors considered. These statistics are uninteresting if considered by themselves, but they lend credence to the results which follow because they show that the experiment was unbiased with regard to background.

4.4 Performance Statistics

A three-way analysis of variance using the approximate method of unweighted means was used to compare performance statistics. The three factors were language, order, and problem. Subjects took both problems, but only in one order using one language. These statistics are found in appendix E.

Subjects using the KAIL-like language on the average answered 16.7 of the 20 questions correctly; subjects using the ALGOL-like language answered only 15.1 correctly. This result is significant at the 2.5 percent level. A chi-squared test validated the normality assumptions. This result supports the hypothesis. The selector construct enabled subjects to answere more questions correctly.

Table 1 shows how each of the four groups performed on the number answered correctly. Group #2, using the KAIL-like language taking the automorphic numbers problem first answered on the average of two more questions correctly than any other group. This result supports the hypotheses explained in section 3.4. Subjects in this group took the arithmetic expression problem after having taken the automorphic numbers problem; and thus, had the advantage of working the program which maximized the use of the selector after they had become familiar with the KAIL-like language.

A typical subject using the KAIL-like language answered 12.3 of the 20 questions correctly on the first try; subjects using the ALGOL-like language answered 11.7. This is not significant. (One might expect significance because appendix F shows a high correlation between number correct and number initially correct, but these means are too close.) It was hypothesized (section 3.4) that group #2 using the KAIL-like language taking the automorphic numbers problem first would perform best. Table 1 shows that they answered about two more questions correctly on the first try than any other group.

An analysis of the average time taken to answer a question correctly showed no significant difference between languages. To obtain these statistics, raw data was first transformed to a normal distribution with mean 0 and standard deviation 1, and an average for each subject on each problem was computed.

	Group			
	#1	#2	#3	#4
Number in group	19	13	15	11
Number answered correctly (20 possible)	15.2	17.5	15.0	15.8
Number answered correctly on the first try (20 possible)	11.7	13.3	11.8	11.4
Standardized mean time to answer correctly	0.055	-0.12	0.14	0.057

	language	problem order
group #1	ALGOL-like	Automorphic Numbers problem first
group #2	KAIL-like	Automorphic Numbers problem first
group #3	ALGOL-like	Arithmetic Expression problem first
group #4	KAIL-like	Arithmetic Expression problem first

Table 1. Group Averages

This removed the effects introduced by differing degrees of difficulty in the questions. Data on questions answered incorrectly was ignored. The standardized mean time to answer a question correctly was 0.097 for the ALGOL-like language subjects and -0.033 for the KAIL-like language subjects. Note that the lower number, even though not significant, means the subjects using the KAIL-like language took less time and therefore did better. Again, the data in Table 1 confirms the hypothesis explained in section 3.4.

Table 2 shows how the average subject in each of the four groups responded to the initial and final self-evaluations. With regard to language effects, only one siginificant result emerged. Disregarding order, the KAIL-like language subjects thought they initially understood the arithmetic expression program at level 3.8 whereas the ALGOL-like language subjects only thought they understood at level 2.7. This is significant at the 2.5 percent level. Across the entire experiment, trends in both the initial and final subjective self-evaluations favored the KAIL-like language; however, no trend was detected when considering the difference between these evaluations.

4.5 Learning Statistics

The experiment measured the time subjects spent learning and reviewing their assigned language. These times are summarized in Table 3. No results were significant. It is interesting, however, to note that the ALGOL-like language subjects spent on the average of 94 seconds longer in learning. This supports the hypothesis that the syntax and semantics of selector construct are at least as easy to learn as the snytax and semantics of the traditional constructs.

4.6 Problem Comparison

Subjects performed better on the arithmetic expression problem than on the automorphic numbers problem. As shown in appendix E, every performance measure except average time to obtain a correct answer exhibited significant

		Automorphic Numbers		Arithmetic Expression	
Number in group		initial	final	initial	final
group #1	19	2.9	4.1	2.9	5.4
group #2	13	2.3	3.6	3.4	5.7
group #3	15	2.0	3.6	2.5	4.5
group #4	11	2.3	3.2	4.2	5.7

Zero represents no understanding; nine represents perfect understanding.
The groups are the same as in Table 1.

Table 2. Self-evaluations

language	number in group	seconds learning	seconds reviewing
ALGOL-like	34	928	20
KAIL-like	24	834	21

Table 3. Average Learning and Reviewing Times

results. Subjects answered more questions correctly ($p<.01$), answered more questions correctly on the first try ($p<.024$), thought the program was easier to understand both before ($p<.01$) and after ($p<.01$) working on the questions, and indicated that they though there was a greater increase in understanding ($p<.025$).

4.7 Correlations

The correlation data supports the model of subject understanding presented in section 3.1. Table 4 shows a high correlation among subjects on the number of questions answered correctly, the number answered correctly on the the first try, and the time taken to answer a question correctly. In answering questions, the model of understanding expected subjects who understood well to perform well and those who understood poorly to perform poorly. Although self-evaluations are fraught with difficulties because of their subjective nature, no unusual correlations appeared. (See appendix F.) This also supports the model of subject understanding.

Between language groups, correlation statistics compared reasonably well except in the area of CS 201 grades. Table 5 shows that better students (measured by class standard score) using the KAIL-like language answered more questions correctly than the poorer students, answered quicker especially on the automorphic numbers problem, gave higher initial self-evaluations, and spent less time learning. Better students using the ALGOL-like language answered about the same number correctly as the poorer students, answered slightly slower, gave slightly lower initial self-evaluations, and spent about the same amount of time learning the language. It is somewhat difficult to interpret these results in terms of the value of the KAIL selector versus the traditional constructs. It is, however, reasonable to infer that better

	ALGOL-like language		KAIL-like language	
factor	AN	AE	AN	AE
NC,NIC	.543	.662	.784	.762
NC,SMTCA	-.424	-.475	-.345	-.381
NIC,SMTCA	-.408	-.486	-.510	-.593

Abbreviations:

AN	Automorphic Numbers Problem
AE	Arithmetic Expression Problem
NC	Number Correct
NIC	Number Initially Correct
SMTCA	Standardized Mean Time to a Correct Answer

Table 4. Correlations with Correct Answer Statistics

Class standard score correlated with:	ALGOL-like language	KAIL-like language
NC on AN	.188	.503
NC on AE	-.083	.327
SMTCA on AN	.233	-.405
SMTCA on AE	.005	.049
ISE on AN	-.236	.354
ISE on AE	.035	.110
TL		-.648

Abbreviations:

ISE	Initial Self-evaluation
TL	Time Learning

See Table 4 for other abbreviations

Table 5. Data Correlated with Class Standard Score

students were able to grasp the newer KAIL-like constructs more easily than poorer students. The selector was initially unfamiliar; the more intelligent students understood the construct quicker and therefore performed better. All subjects were reasonably familiar with the ALGOL-like language constructs before the experiment began. Since most of the questions were relatively simple, poorer students, drawing from their background, were able to perform as well as better students.

5. CONCLUSION

5.1 Summary of Results

Subjects using the KAIL-like language answered more questions correctly than subjects using the ALGOL-like language. Disregarding order, they also though that they initially better understood the arithmetic expression program. These results were significant.

Of the four groups, the group that performed best used the KAIL-like language and took the arithmetic expression problem last. This group worked the program which maximized the use of the selector after having become familiar with the KAIL-like language. It was hypothesized that these subjects would perform best; and they did.

Statistics on the number of questions initially answered correctly, average time taken to obtain a correct answer, and initial and final self-evaluations all exhibited trends toward the KAIL-like language. No performance statistics favored the ALGOL-like language.

The results supported the basic hypothesis. Programmers read and understood the KAIL selector more easily than an equivalent set of traditional language constructs.

5.2 Suggestions for Further Research

The results indicate that further research is likely to be fruitful. This experiment tested for the subject's ability to read, understand, and to some extent to learn the syntax and semantics of the KAIL selector. Other experiments should be conducted to test a programmer's ability to write, debug, and modify.

Since the experiment used CS 201 students as subjects, the results are valid for this category; but other programmers may perform differently. Experiments using a wide variety of subjects are likely to be fruitful.

In order to administer this experiment, some automated methodologies for psychological experiments in computer programming were developed. These should be refined, and others should be developed. Because more and better data can be gathered and because computer resources can be exploited, automated methods are likely to be useful for experiments in computer programming.

REFERENCES

[1] Sammet, Jean E., Programming Languages: History and Fundamentals, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1969.

[2] Schneiderman, Ben, "Experimental Testing in Programming Languages, Stylistic Considerations, and Design Techniques, " Technical Report No. 16, Indiana University, Computer Science Department, October, 1974.

[3] Embley, David W. and Wilfred J. Hansen, "The KAIL Selector - A Unified Control Construct," (to be published in SIGPLAN Notices).

[4] Embley, David W., "An Introduction to KAIL," Lesson Kaids on the PLATO System, University of Illinois, August, 1975.

[5] Nievergelt, J., E. M. Reingold, and T. R. Wilcox, "The Automation of Introductory Computer Science Courses," SIGCUI Bulletin, Vol. 8, No. 1, 1974, pp. 15-21.

[6] Alpert, D. and D. L. Bitzer, "Advances in Computer Based Education," Science 167, 1970, pp. 1382-1590.

[7] Sherwood, Bruce A., The TUTOR Language, CERL, University of Illinois, June 1974.

[8] Weinberg, Gerald M., The Psychology of Computer Programming, Van Nostrand Reinhold Company, New York, 1971.

[9] Sime, M. E., T. R. G. Green, and D. J. Guest, "Psychological Evaluation of Two Conditional Constructions Used in Computer Languages," International Journal of Man-Machine Studies, Vol. No. 1, 1973, pp 105-113.

[10] Weissman, Laurence M., "Psychological Complexity of Computer Programs: An Experimental Methodology," SIGPLAN Notices, Vol. 9, No. 6, June, 1974, pp. 25-36.

[11] Weissman, Laurence M., "A Methodology for Studying the Psychological Complexity of Computer Programs," Technical Report CSRG-37, University of Toronto, August, 1974.

[12] Schneiderman, Ben, and Mao-Hsian Ho, "Two Experiments in Programming Behavior," Technical Report No. 17, Indiana University, Computer Science Department, October, 1974.

[13] Gannon, John D., "Language Design to Enhance Programming Reliability," Technical Report CSRG-47, University of Toronto, January, 1975.

[14] Knuth, D. E., "Structured Programming with goto Statements," ACM Computing Surveys, Vol. 6, No. 4, December, 1974, pp. 261-301.

[15] Dykstra, E. W., "EWD316: A Short Introduction to the Art of Programming," Technical University, Eindhoven, The Netherlands, August, 1971.

APPENDIX A

FULL GRAMMAR OF THE KAIL SELECTOR

```
selector :   selection
             | seltag "_" selection "_" seltag
             | "*" selection "*"
             | seltag "_" "*" selection "*" "_" seltag
selection :  "[" statement-sequence [control  choices]  "]"
control :    type [expression]
type :       "if" | "while" | "until"
choices :    λ| "|" [comparand] statement-sequence [action] choices
comparand :  [relation] expression ":" | "|" "else" ":"
action :     "exit" [seltag] | "again" [seltag]
expression-selector : expression-selection
             | seltag "_" expression-selection "_" seltag


expression-selection :  "[" statement-sequence control  expression-choices "]"
expression-choices :  "|" [comparand] expression
             | "|" [comparand] expression  expression-choices
```

Restrictions:

1. if may only be used with [...],
2. while and until may only be used with *[...]*,
3. seltags must match.

APPENDIX B

SUBJECT BACKGROUND DATA

The table on background information is understood as follows:

Group:

	language	problem order
group #1	ALGOL-like	Automorphic Numbers problem first
group #2	KAIL-like	Automorphic Numbers problem first
group #3	ALGOL-like	Arithmetic Expression problem first
group #4	KAIL-like	Arithmetic Expression problem first

Education Level:

1. Freshman
2. Sophomore
3. Junior
4. Senior
5. Four years plus, but no degree
6. One year graduate school
7. Two years graduate school
8. Three years graduate school
9. Four or more years graduate school

Language Experience:

1. Never heard of it
2. Knew of its existence
3. Knew a little about it or had written a program in it
4. Knew it fairly well or had written several programs in it
5. Knew practically everything about it or had programmed extensively using it

PLATO Experience:

$$100 \times \frac{\text{subjects with PLATO experience}}{\text{number in group}}$$

Class Standard Score in CS 201:

mean:	500
standard deviation:	170

	Group			
	#1	#2	#3	#4
Number in Group	20	17	20	15
Education Level	2.8	2.3	3.1	2.7
Language Experience	2.2	2.2	2.1	2.2
ALGOL 68	2.1	1.8	1.9	1.8
ALGOL W	1.5	1.6	1.8	1.8
COBOL	2.3	2.2	2.3	2.3
EULER	1.2	1.2	1.4	1.0
FORTRAN	3.7	3.8	3.6	3.7
KAIL	1.1	1.2	1.1	1.0
LISP	1.4	1.2	1.3	1.4
PL/1	4.4	4.5	4.2	4.5
SNOBOL	1.6	1.7	1.8	1.8
TUTOR	2.4	2.6	2.1	2.2
PLATO Experience	75%	65%	60%	87%
Class Standard Score	489	517	512	513

APPENDIX C

PROBLEMS AND PROGRAM SOLUTIONS

Automorphic Numbers

Problem

This program finds all automorphic numbers which are less than or equal to the number read at the beginning of the program. A number, n, represented by $d_1d_2 \ldots d_k$ is automorphic if the last k digits of n^2 are $d_1d_2 \ldots d_k$.

Some examples of automorphic numbers are.

5 6 25

As you can see, the squares of these numbers end with the digits which represent the original number.

$2\underline{5}$ $3\underline{6}$ $6\underline{25}$

ALGOL-like Solution

```
integer i,j,k,n,isqr;
read n;
i ⇐ 1;  k ⇐ Ø;
while i ≤ n do
    isqr ⇐ i×i;
    j ⇐ 1;
    while j ≤ i do j ⇐ j×1Ø; od;
    if i=(isqr - j×(isqr÷j)) then write i,isqr; fi;
    case k
        :₀ i ⇐ i + 4;  k ⇐ 1;
        :₁ i ⇐ i + 1;  k ⇐ 2;
        :₂ i ⇐ i + 5;  k ⇐ Ø;
    esac;
od;
```

KAIL-like Solution

```
integer i,j,k,n,isqr;
read n;
i ⇐ 1;  k ⇐ Ø;
*[ i ≤ n ▌
    isqr ⇐ i×i;
    j ⇐ 1;
    *[ j ≤ i ▌ j ⇐ j×1Ø; ]*;
    [ i=(isqr - j×(isqr÷j)) ▌ write i,isqr; ];
    [ k
        ▌₀ i ⇐ i + 4;  k ⇐ 1;
        ▌₁ i ⇐ i + 1;  k ⇐ 2;
        ▌₂ i ⇐ i + 5;  k ⇐ Ø;
    ];
]*;
```

Arithmetic Expressions

Problem

This program analyzes arithmetic expressions involving the symbols "+", "-", and "a". It determines whether an expression is syntactically correct, deletes unnecessary symbols, and outputs the result.

Thus, for example, the expression "-+a-+-a+-a" becomes "-a+a-a", and the string "-a-" is in error.

To make the program easier, the input is encoded as:

"+"	=	1
"-"	=	2
"a"	=	3

The last input is always a Ø denoting end of file (eof).

Thus, the above expression "-+a-+-a+-a" is encoded for input as "2 1 3 2 1 2 3 1 2 3 Ø".

ALGOL-like Solution

```
comment  Ø = eof, 1 = "+", 2 = "-", 3 = "a";

integer i,state;
character prfx;
state ← 1;
while state ≥ Ø do
    case state
        :0 write "e","r","r","o","r";
           state ← -1;
        :1 prfx ← " "; read i;
           state ← case i :0 Ø :1 1 :2 2 :3 3 esac;
        :2 prfx ← "-"; read i;
           state ← case i :0 Ø :1 2 :2 1 :3 3 esac;
        :3 write prfx,"a"; read i;
           state ← case i :0 -1 :1 4 :2 5 :3 Ø esac;
        :4 prfx ← "+"; read i;
           state ← case i :0 Ø :1 4 :2 5 :3 3 esac;
        :5 prfx ← "-"; read i;
           state ← case i :0 Ø :1 5 :2 4 :3 3 esac;
    esac;
od;
```

KAIL-like Solution

```
comment  Ø = eof, 1 = "+", 2 = "-", 3 = "a";

integer i,state;
character prfx;
state ⇐ 1;
*[ state
    ▮0 write "e","r","r","o","r";
         state ⇐ -1;
    ▮1 prfx ⇐ " ";  read i;
         state ⇐ [ i ▮0 Ø ▮1 1 ▮2 2 ▮3 3 ];
    ▮2 prfx ⇐ "-";  read i;
         state ⇐ [ i ▮0 Ø ▮1 2 ▮2 1 ▮3 3 ];
    ▮3 write prfx,"a";  read i;
         state ⇐ [ i ▮0 -1 ▮1 4 ▮2 5 ▮3 Ø ];
    ▮4 prfx ⇐ "+";  read i;
         state ⇐ [ i ▮0 Ø ▮1 4 ▮2 5 ▮3 3 ];
    ▮5 prfx ⇐ "-";  read i;
         state ⇐ [ i ▮0 Ø ▮1 5 ▮2 4 ▮3 3 ];
]*;
```

APPENDIX D

QUESTIONS

Automorphic Numbers Problem

1. Each time the write statement is executed, how many values are output?
2. How many times does k appear on the left hand side of "<="?
3. List the values that k can have during program execution. (Separate values with "," or space.)
4. How many times does i appear on the left hand side of "<="?
5. List, in order, the first seven values that i can have. (Separate values with "," or space.)
6. List, in order, the first four values that isqr can have. (Separate values with "," or space.)
7. If i is 1537, what value does j have when j *(isqr ÷ j) is evaluated?
8. If i is 5, what is (isqr - j*(isqr ÷ j))?
9. If n is 52, how many times is the statement "j<=1" executed?
10. Approximately what is the probability that i is equal to n the last time through the outer loop?

Arithmetic Expression Problem

1. How many read statements are in the program?
2. How many times does state appear on the left hand side "<="?
3. In how many different statements is "-" assigned to prfx?
4. In how many different statements is "+" assigned to prfx?
5. If the input string consists only of "0" what is the output?
6. If the input string is "3 0", what is the output?

7. A sequence of two statements can be simplified to "read state". What statement immediately precedes these?
8. Which statement can be replaced by "state <= -13" without altering the program behavior?
9. When the input is "1 2 3 3 0", what sequence of characters is output?
10. What is the sequence of state values for the input "2 3 1 3 0"? (Separate values with "," or space.)

APPENDIX E

PERFORMANCE STATISTICS

A three-way analysis of variance using the approximate method of unweighted means was used to compute these statistics. The three factors were language, order, and problem. Subjects took both problems, but only in one order using one language.

Number of questions answered correctly

Source of variance	df	MS	F	significance
language	1	17.24	5.34	<2.5%
order	1	6.49	2.01	
language x order	1	3.97	1.23	
subjects within levels of language and order (error)	54	3.22		
problem	1	14.63	13.55	<1%
language x problem	1	2.45	2.27	
problem x order	1	0.08	0.08	
language x problem x order	1	2.62	2.43	
subjects within levels of language and order x problem (error)	54	1.08		

Number of questions answered correctly on first attempt

Sources of varience	df	MS	F	significance
language	1	2.40	0.48	
order	1	5.91	1.19	
language x order	1	6.05	1.22	
subjects within levels of language and order (error)	54	4.97		
problem	1	10.68	5.64	<2.5%
language x problem	1	0.11	0.06	
problem x order	1	0.34	0.18	
language x problem x order	1	0.10	0.05	
subjects within levels of language and order x problem (error)	54	1.89		

Average time taken to answer a question correctly

To obtain these statistics, raw data on the time taken to answer a question correctly was first transformed to N(0,1), and an average for each subject on each problem was computed. Data on questions answered incorrectly was ignored.

Sources of variance	df	MS	F	significance
language	1	0.47	1.34	
order	1	0.49	1.39	
language x order	1	0.06	0.17	
subjects within levels of language and order (error)	54	0.35		
problem	1	0.00	0.00	
language x problem	1	0.03	0.18	
problem x order	1	0.31	1.94	
language x problem x order	1	0.13	0.82	
subjects within levels of language and order x problem (error)	54	0.16		

Initial self-evaluation

Sources of variance	df	MS	F	significance
language	1	1.57	0.31	
order	1	2.12	0.41	
language x order	1	3.19	0.62	
subjects within levels of language and order (error)	54	5.13		
problem	1	15.84	7.98	<1%
language x problem	1	4.93	2.48	
problem x order	1	1.30	0.65	
language x problem x order	1	0.39	0.19	
subjects within levels of language and order x problem (error)	54	1.99		

Final self-evaluation

Sources of variance	df	MS	F	significance
language	1	1.47	0.18	
order	1	24.61	3.02	
language x order	1	0.81	0.10	
subjects within levels of language and order (error)	54	8.15		
problem	1	74.29	31.43	< 1%
language x problems	1	0.33	0.14	
problem x order	1	0.08	0.03	
language x problem x order	1	0.69	0.29	
subjects within levels of language and order x problem (error)	54	0.24		

Difference between initial and final self-evaluation

Sources of variance	df	MS	F	significance
language	1	0.00	0.00	
order	1	12.30	1.60	
language x order	1	0.78	0.10	
subjects within levels of language and order (error)	54	7.66		
problem	1	21.52	7.04	< 2.5%
language x problem	1	2.72	0.89	
problem x order	1	0.75	0.24	
language x problem x order	1	0.04	0.01	
subjects iwthin levels of language and order x problem (error)	54	3.05		

APPENDIX F

CORRELATIONS

A missing data correlation program was used to compute these statistics. In the following:

AN = Automorphic Numbers Problem,

AE = Arithmetic Expression Problem,

* = significant with probability .010,

** = significant with probability .005.

Correlation factors:

1. Number correct (AN)
2. Number correct (AE)
3. Number initially correct (AN)
4. Number initially correct (AE)
5. Standardized mean time to a correct answer (AN)
6. Standardized mean time to a correct answer (AE)
7. Initial self-evaluation (AN)
8. Initial self-evaluation (AE)
9. Final self-evaluation (AN)
10. Final self-evaluation (AE)
11. Difference in self-evaluations (AN)
12. Difference in self-evaluations (AE)
13. Language knowledge
14. Class standard score
15. Time spent learning

Correlations for subjects using the ALGOL-like language

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1.															
2.	.528**														
3.	.543**	.421**													
4.	.293	.662**	.409*												
5.	-.424**	-.310	-.408*	.085											
6.	-.333	-.475**	-.183	-.486**	.371										
7.	.183	.383*	-.002	.186	-.342	-.425**									
8.	.319	.156	.209	.072	-.288	-.074	.508**								
9.	.640**	.523**	.526**	.405*	-.220	-.347	.463**	.278							
10.	.363	.380	.258	.392*	-.087	-.229	.277	.265	.549**						
11.	.476**	.174	.531**	.237	.093	.042	-.454**	-.188	.579**	.298					
12.	.126	.256	.102	.322	.111	-.167	-.079	-.414*	.328	.768**	.405*				
13.	.133	.265	.204	.384*	-.175	-.370	.515**	.178	.425**	.294	-.046	.160			
14.	.188	-.083	.050	.021	.233	.005	-.236	-.072	.031	.007	.296	.055	-.016		
15.	-.326	-.287	.082	-.043	.240	.356	-.492**	-.435**	-.286	.060	.165	.346	-.106	.035	

Correlations for subjects using the KAIL-like language

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1.															
2.	.566**														
3.	.784**	.499**													
4.	.623**	.762**	.519**												
5.	-.345	-.237	-.510**	-.265											
6.	-.426	-.381	-.195	-.593**	.332										
7.	.331	.228	.077	.328	-.019	-.241									
8.	.228	.117	.039	.172	.229	-.015	.371								
9.	.727**	.410	.482*	.543**	-.090	-.219	.486*	.369							
10.	.506**	.228	.479**	.303	.028	.160	.106	.557**	.600**						
11.	.516**	.249	.461*	.308	-.082	-.035	-.304	.088	.686**	.553**					
12.	.336	.150	.473**	.183	-.169	.197	-.211	-.312	.313	.615**	.507**				
13.	.184	.209	.063	.106	-.325	-.044	.066	-.062	-.063	-.159	-.123	-.122			
14.	.503**	.327	.287	.117	-.405	.049	.354	.110	.226	.014	-.068	-.087	.434		
15.	.249	-.065	-.073	-.131	.217	.332	-.188	-.302	-.107	.120	.054	.405	-.267	-.648**	

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-75-759	2.	3. Recipient's Accession No.
4. Title and Subtitle AN EXPERIMENT ON A UNIFIED CONTROL CONSTRUCT			5. Report Date August 1975
			6.
7. Author(s) David W. Embley			8. Performing Organization Rept. No. UIUCDCS-R-75-759
9. Performing Organization Name and Address Department of Computer Science University of Illinois at Urbana-Champaign Urbana, Illinois 61801			10. Project/Task/Work Unit No.
			11. Contract/Grant No. US-NSF-EC-41511
12. Sponsoring Organization Name and Address National Science Foundation Washington, D.C.			13. Type of Report & Period Covered Technical Report
			14.

15. Supplementary Notes

16. Abstracts

This report describes and gives the results of an experiment designed to investigate the psychological soundness of a proposed unified control construct, the KAIL selector. The KAIL selector subsumes several traditional control constructs including if-then-else, case, and while. The experiment compared two sets of control constructs to determine their effect on understanding. One set consisted of the traditional if-then-else, case, and while; the other set consisted of a simplified KAIL selector. Results showed that subjects understood programs better when they were written using the KAIL selector.

17. Key Words and Document Analysis. 17a. Descriptors

Programming Languages
Programming Language Design
Psychological Experiments in Programming
Computer-Assisted Instruction

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement Release Unlimited	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 64
	20. Security Class (This Page UNCLASSIFIED	22. Price --

INSTRUCTIONS FOR COMPLETING FORM NTIS-35 (10-70) (Bibliographic Data Sheet based on COSATI Guidelines to Format Standards for Scientific and Technical Reports Prepared by or for the Federal Government, PB-180 600).

1. **Report Number.** Each report shall carry a unique alphanumeric designation. Select one of the following types: (a) alphanumeric designation provided by the sponsoring agency, e.g., **FAA-RD-68-09**; or, if none has been assigned, (b) alphanumeric designation established by the performing organization e.g., **FASEB-NS-87**; or, if none has been established, (c) alphanumeric designation derived from contract or grant number, e.g., **PH-43-64-932-4.**

2. Leave blank.

3. **Recipient's Accession Number.** Reserved for use by each report recipient.

4. **Title ond Subtitle.** Title should indicate clearly and briefly the subject coverage of the report, and be displayed prominently. Set subtitle, if used, in smaller type or otherwise subordinate it to main title. When a report is prepared in more than one volume, repeat the primary title, add volume number and include subtitle for the specific volume.

5. **Report Dote.** Each report shall carry a date indicating at least month and year. Indicate the basis on which it was selected (e.g., date of issue, date of approval, date or preparation.

6. **Performing Orgonizotion Code.** Leave blank.

7. **Author(s).** Give name(s) in conventional order (e.g., John R. Doe, or J.Robert Doe). List author's affiliation if it differs from the performing organization.

8. **Performing Orgonizotion Report Number.** Insert if performing organization wishes to assign this number.

9. **Performing Orgonizotion Nome ond Address.** Give name, street, city, state, and zip code. List no more than two levels of an organizational hierarchy. Display the name of the organization exactly as it should appear in Government indexes such as **USGRDR-I.**

10. **Project/Tosk/Work Unit Number.** Use the project, task and work unit numbers under which the report was prepared.

11. **Controct/Grant Number.** Insert contract or grant number under which report was prepared.

12. **Sponsoring Agency Nome ond Address.** Include zip code.

13. **Type of Report ond Period Covered.** Indicate interim, final, etc., and, if applicable, dates covered.

14. **Sponsoring Agency Code.** Leave blank.

15. **Supplementory Notes.** Enter information not included elsewhere but useful, such as: Prepared in cooperation with . . . Translation of . . . Presented at conference of . . . To be published in . . . Supersedes . . . Supplements . . .

16. **Abstroct.** Include a brief (200 words or less) factual summary of the most significant information contained in the report. If the report contains a significant bibliography or literature survey, mention it here.

17. **Key Words ond Document Anolysis. (o). Descriptors.** Select from the Thesaurus of Engineering and Scientific Terms the proper authorized terms that identify the major concept of the research and are sufficiently specific and precise to be used as index entries for cataloging.
   (b). Identifiers ond Open-Ended Terms. Use identifiers for project names, code names, equipment designators, etc. Use open-ended terms written in descriptor form for those subjects for which no descriptor exists.
   (c). COSATI Field/Group. Field and Group assignments are to be taken from the 1965 COSATI Subject Category List. Since the majority of documents are multidisciplinary in nature, the primary Field/Group assignment(s) will be the specific discipline, area of human endeavor, or type of physical object. The application(s) will be cross-referenced with secondary Field/Group assignments that will follow the primary posting(s).

18. **Distribution Stotement.** Denote releasability to the public or limitation for reasons other than security for example "Release unlimited". Cite any availability to the public, with address and price.

19 & 20. **Security Clossificotion.** Do not submit classified reports to the National Technical Information Service.

21. **Number of Poges.** Insert the total number of pages, including this one and unnumbered pages, but excluding distribution list, if any.

22. **Price.** Insert the price set by the National Technical Information Service or the Government Printing Office, if known.